Issue 26 | Feb 2009

Blender learning made easy

blender art MAGAZINE

Blender & Gaming

Going Retro With the BGE

Blender in 2.5D Games

Shootin' Annie

Mii School

COVERART Zero Ballistics -by Bernhard Bauer

blender art MAGAZINE

EDITOR
Gaurav Nawani gaurav@blenderart.org

MANAGING EDITOR
Sandra Gilbert sandra@blenderart.org

WEBSITE
Nam Pham nam@blenderart.org

DESIGNER
Gaurav, Sandra, Alex

PROOFERS
Brian C. Treacy
Bruce Westfall
Daniel Hand
Daniel Mate
Henriël Veldtmann
Joshua Leung
Joshua Scotton
Kevin Braun
Mark Warren
Noah Summers
Patrick ODonnell
Phillip
Ronan Posnic
Scott Hill
Wade Bick
Valérie Hambert

WRITERS
Rosario Azzarello
Raul Gonzalez Luy
Alex Aylesbury
Paul Virapen
Hubert Gizeskowiak (PM)
Gaurav Nawani
Tony Mullen
Moisés Espínola

COVER ART
Zero Ballistics - by Bernard Bauer

CONTENTS

Sandra Gilbert
Managing Editor

"Not surprisingly, many developers are turning to Blender for asset production as well as rapid prototyping... "

Look anywhere on the net, and you will notice that creating games is becoming as popular as playing them. Whole sub-cultures are springing up around online games and the developers that create them. I even have to admit that I seem to have gotten hooked on a few myself.

Now obviously, not everyone is using the same software or tools in their game production pipelines. And honestly, trying to compile a comprehensive list would be a daunting task indeed.

But once the coding is ironed out, one common issue all developers run into, is how and what to use to create their game assets. Not surprisingly, many developers are turning to Blender for asset production as well as rapid prototyping.

In this issue we get to hear how several developers and game companies are using Blender to create really fun and interesting games.

So get ready for some serious gaming fun.
sandra@blenderart.org

So, short of being the ultimate game creation wizard, what are you going to do?

Introduction

Lately, creating games has become as popular as playing them. Considering the number of available game engines to be found these days, the possibilities for efficient work flows are endless. But no matter what style or type of game you choose to create or engine you use, there are some common elements that you are going to need.

Creating games requires a lot of time and effort. Depending on the game engine you use, there will probably be a fair amount of coding involved. If you write your own game engine, there will be even more coding to deal with.

This, of course, leaves less time for creation of game assets. Even with the best coding and game play, lack of actual game assets is going to sink your game.

So, short of being the ultimate game creation wizard, what are you going to do?

You are going to check out the following links and get yourself some assets, or at the very least a starting point that you can work from and embellish as you go.

http://www.katsbits.com/

Low poly modeling tutorials, tips and tricks.

http://opengameart.org/

Free game content.

Game texture database.

Free for download most come with normal maps.

CGTUTS

Creating a rocky video game terrain in Blender.

SemiPro Tutorials

Model & Texture a Low Poly Race Track.

http://blender-games.com/

Website dedicated to blender games, great place to get ideas and play a game or two.

Blender 2.50 Alpha 2

Another alpha test build has been released to help ensure that bug reporters have a good reference for ongoing testing and reporting. There have been numerous (read that to be hundreds and then some) bug fixes and lots of improvements and updates to missing and incomplete features. While there are still a few features not fully implemented, there is more than enough for some serious fun and testing.

Check out the 2.50 Alpha 1 release log for more details of features and what to expect in this release.

DVD training 5: Chaos & Evolutions

Digital painting course using free/open source software.

Pre-order discount of 10%. DVD expected to ship mid March.

This DVD covers all aspects of creating advanced digital paintings, concept art, character design, creature design, environments, illustrations and model sheets. Seven main tutorial videos are devoted to teach the best tips and tricks about digital sketching, drawing, Alchemy, coloring, painting, finishing and creating model sheets. And eight additional videos will present you with more than 20 hours of full time lapse in HD (1440x896) minimally commented so that you are free to explore and learn the part you are interested in.

The DVD assumes you already have a tablet and know the basics of drawing from imagination, 2D painting software and image manipulation. It is especially targeted at people who look for a strong work flow to create a 2D project from A to Z in a collaborative or professional environment.

Chaos & Evolutions includes Alchemy and GIMP 2.6.6 patched with GIMP-painter for Windows, Mac OSX and Linux, as well as a customized brush kit, and all WIP files. It also includes MyPaint for Windows and Linux. All the examples have been done with only free and open source software, mainly GIMP. You can follow the tutorials with your own 2D application too; like Photoshop / Painter / TVPaint / Krita / etc... (Note: one bonus video uses Mypaint!).

The DVD tutorials use a music soundtrack without voice over. Unobtrusive, simple English labels will guide you, which makes it especially targeted at an international public.

The videos are in 1440x896 and play well in the free VLC player.

Announcing the BlenderNewbies Blender 2.5 Training DVD Pre - Order

Kernon has announced the pre-order of his new upcoming training DVD. He will be covering Blender 2.5 exclusively and it will contain many hours of new and unpublished tutorials.

There won't be any 2.5 versions of any of his existing tutorials so this will be fresh new content!

The training will be mostly small, project-based tutorials so that what you learn is in the context of a real project scenario. This helps you understand not only How to do something, but also helps you have a better understanding of Why and When. Also, the project-based tutorials will allow you to repeatedly experience the full work flow so you develop a better understanding of how things work together.

For further details and ordering info, stop by blendernewbies.blogspot.com

Circolino

By Rosario Azzarello

Introduction

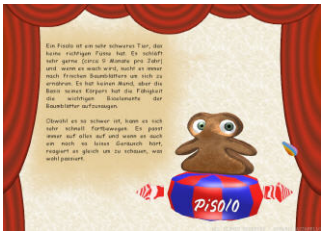
Circolino is a bundle of five Serious Games and self-built input devices to play games while making a range of different physiotherapy exercises. Every time the children play Circolino, the system logs very important information for the therapy and then saves it in a well formatted CSV file to be used for analyzing the progress of every single patient.

This Serious Game was made for the Children's Rehabilitation Center of Affoltern (Switzerland) and is adapted to children under 12 years old with motor function problems.

Playing is something that the human races has always done. For this reason playing games is a very important part in the life of every of us. Games are part of our culture as well, regardless of which game is being played.

Sadly, not all people are able to play video games. There are people that have problems with motor functions and most of these are children. Normally, physiotherapy involves helping children achieve their gross motor functions and enabling them to develop their physical independence. The Physiotherapists assess, design and carry out stimulation exercises for children to improve their physical skills, and to prevent further deformities and minimize the effects of disability.

Helping children with physiotherapy is quite different than helping adults, from an anatomical, physiological and psychological point of view. To treat children effectively all these issues need to be considered.



My first step to create this game was to analyze, together with a Physiotherapist, the different exercises that these children executed every day. After that I collected all the important information and wrote the GDD (Game Design Document). While writing it, I paid attention to all aspects of the game with the 'eyes' of these children.

It wasn't as easy as we can imagine. Most of these children have motor problems not only because the muscles are not working well, but also because the brain has some problems, too.

By Rosario Azzarello

For this reason, I wanted to create mini-games that help to train the different parts of the brain and not only the muscles: they must work together.

The mini-games of Circolino train these parts of the brain that are used to memorize, to calculate, to recognize, to see and to hear: parietal lobe, frontal lobe, temporal lobe and occipital lobe.

The self built input devices train the motor functions while playing these mini-games. I've created something that has this 'Wii style' thing going on and giving a whole different gaming experience than just ordinary keyboard or mouse. Most of these children can't hold any controller in their hands, just because they have problems to move it, too. My objective was to create something to wear, not so heavy, easy to install and

to use. To use these input devices we need to just plug-in the USB cable and we are ready to play: no extra drivers to install and compatible with most operating systems.

Last but not least: the children love Circolino!

I was very happy to use my knowledge to create a game that is focusing on a target audience that is not always considered in the game industry.

To create Circolino I used Blender to create the game and the 3D assets (environment, characters and items), Python for scripting it, Gimp for the textures, Inkscape for the 2D game elements, Audacity for the Sounds and the Music, and Scribus for the documentation.

Rosario Azzarello

Porfolio: http://www.gamedesign.ch

Video: http://www.youtube.com/watch?v=zRqbJbBIcfQ

Retro With the BGE

By Raul Gonzalez Luy

Introduction

With the newest consoles generation and all those speed ups and new graphical features (2D screen effects, GLSL, softbodies, etc.) that came with the BGE, some older ways of gaming experience seem to fade away from the memory of both gamers and game-creators. Naturally, none of us would like to return to the ages of Atari, but there are some stuff from game history that are still nice, if used correctly. This article focuses on 2D sprites.

To make it shorter and not dangle too much in history of video games, I'll say that the use of 2D characters on a 3D background has been around since the very beginning, from Wolfenstein 3D and Doom and the several clone games they had. In the early 90s, the famous "Mode7" in the SuperNintendo allowed 3D deformation of a plane, which was used a lot for making racing games (Mario Kart for example), naturally those cars were 2D.

This technique was most notably used in the era of Playstation 1 and Nintendo 64, due to the lack of graphical power those consoles had. In one hand they could have several polygons on screen textured and shaded, but in the other, not enough to have nicely detailed personages; almost, box looking characters with heavily pixelated textures check image of Vagrant Story for PSX, a good looking PSX game of 1998.

While not because of that, this style of graphics has not disappeared yet, we can still find it on Nintendo DS, Playstation Portable, Wii and even Playstation 2 and 3. Communication.

Examples:

Famous games using this technique are: Doom (PC), Wolfenstein3D, Super Mario Kart (SuperNes), F-Zero (SuperNes), Xenogears (PSX), Breath of Fire(PSX), Grandia (PSX), Ragnarok Online (PC), Disgaea (PS2, and PS3), Mario Kart 64 (N64), Paper Mario (N64,GameCube, and WII), etc.

Following the same idea, let's start with my demo:For this, it's required to have some advanced knowledge on the Game Engine and python.

Xenogears (PSX):

Ragnarok OnRagnarok Online (PC):

Breath Of Fire (PSX)

By Raul Gonzalez Luy

For now the easiest thing to do is take the demo and replace the Sprite sheets and the scenery.

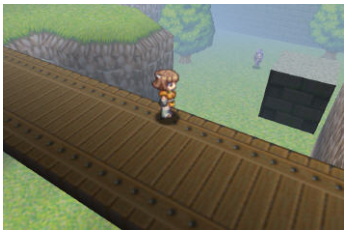
In the future I'm planning on making an in-depth tutorial, when the script becomes more friendly to users.

Disgaea 3 (PS3)

The theory, at least, is simple:

All that is needed is already inside the demo file and working. The link to download the file is at the end of the article.

Paper Mario Wii:

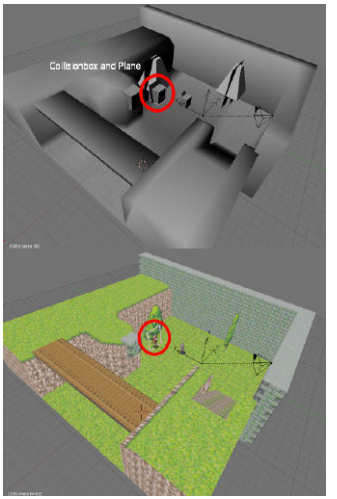


You need a 3D background, an invisible collision box that has a plane that tracks the camera parented to it, which will be the character. But how does the character's drawing change when the camera rotates around him? Well, there is where my script comes in.

The scripts at a glance:

It ain't easy to setup (I'm pretty sure the BGE wasn't made thinking on this, hehe), since you need 3 scripts for this:

- "holiday": The first script wasn't made by me (and I hardly understand it, updating it to work on blender2.49 was incredibly hard). It was made by Doc Holyday (don't know true name) and I use it here to load a Character sprite sheet and have it ready for animation, and animate it.
- "CGLHoliday": The second, by me, is used to activate the animations using the arrows keys.
- "angle3": The third one, also by me, computes the angles of the camera using the world as reference, the character in reference with the world and the character in reference with the camera. In other words, a tough job.



- The other 3 scripts ("holiday, "CGLHolidayNPC", and "angleNPC" for other characters) are made for non-playable-characters (as town people for example). They were made to automatically animate and change the orientation of the character based on its movement, for example following an IPO, as the file, or controlled by an AI script.

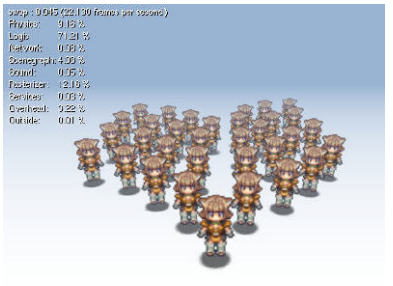
Finally you need a character sprite sheet. Remember, the BGE works with textures that are power of 2 (for example 2x2, 4x4, 8x8, 16x16, 32x32,....1024x1024 is the maximum I think). Be careful on the UVMapping, it's vital, follow the demo file as reference:

By Raul Gonzalez Luy

Issues:

The fact that you need 3 scripts per character is still a problem to solve (it makes it hard to use). Also, I would like to add the possibility of diagonals on the character orientation (or at least have it ready as an option between 4 or 8 directions). But having it working and playable was so much fun when I made the first version!!

Still, this results in pretty good performance. On an old Athlon XP +2600 with a Geforce FX5200 and 1.3 Gigabytes of RAM memory, you can control up to 33 characters simultaneously, all of them animated, with collisions and controlled by the player. As the picture shows, like a little army (hahaha).



Links:

Here are links to my gallery on YouTube where a video of this working is shown (and some others also of the BGE or renders), a link were the files are located for download, a link to the old thread at BlenderArtist where I updated with a recent post, and a link to my deviantArt page were there is also some information in the Gallery.

http://www.youtube.com/user/RolazoGL

http://sites.google.com/site/bgeatelier/

http://blenderartists.org/forum/showthread.php?t=144683

http://rologl.deviantart.com/

2.5D Games

By Alex Aylesbury

Introduction

At Matica.com we develop a varied mix of games and animations. From 2D games using cartoon style vector graphics to real time 3D games and the full gamut in between.

The production pipeline for our 3D games is probably no different to most 3D game development. Character design, high poly modeling, texturing, rigging, animation, effects, low poly texture baking and normal maps, etc.

What seems to be covered less is how you can include 3D tools such as Blender in your 2D or 2.5D games and animation production pipeline.

You can produce just as impressive graphical assets using Blender with your 2D or 2.5D games as you can with traditional pixel based and vector based imaging tools. With the added benefit of being able to make significant changes to your scenes and render a new result almost instantly.

To illustrate this I'll outline how we have used Blender in our production pipeline for some of our Flash games. We have used Blender extensively throughout our catalog so will limit the scope to 3 projects; namely 'Matica Air Race', '6 Nations Rugby' and 'Winter Pinball'.

Winter Pinball

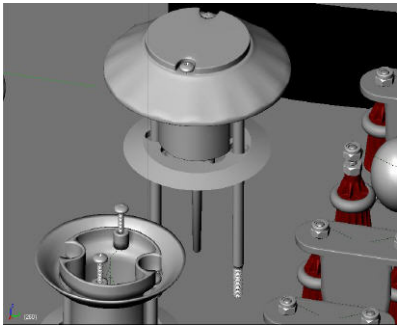
The simplest use was for the playfield background of our Winter Pinball Game. All that was needed was a 2D image of a 3D Wintry scene, trees, snow, and some stars. We could have used Photoshop, Gimp, Inkscape, Flash or Illustrator but it only took about 1 hour to create using Blender.

A subdivided plane for the landscape, a simple particle system for the snow, and 2 very simple subsurfed meshes for the trees and stars took minutes to create. Then all that was left to do was to play around with the colour ramps and material nodes to get the look we were after.

During the game development process we changed the layout of the pinball table and were relieved the playfield scene was created using Blender. The trees in the scene overlapped table components which had an undesired effect on the overall look. The fix was just a case of rearranging the scene a little to fit the new layout and then rendering it out again.

The rest of the pinball table was modeled in detail using Blender. We needed a 3D model of a pinball table for another couple of projects so had to go through the process anyway. It would have been just as quick to hand draw the table, but having it in 3D allows us to make changes to the table and have our new image at the push of a button.

The specular highlights and shadows help a lot with the 3D look but were not enough on their own. For the final render we tilted the table and used an orthographic camera.



The tilt was needed to help the 3D look (showing the front edge of the table components), and we needed everything to be orthographic for the graphics to match up to the underlying game physics keeping the sides of the table parallel. Render layers allowed us to render everything below the ball and everything above the ball as separate images. This way in the final game we could have the ball passing over and under the correct table items adding to the realism.

We tried using a separate shadow layer to add to the realism. It made just a subtle difference as it only effected shadows on the ball which was moving so quickly you could hardly notice them. It would add to the final file size of the game so we went with just 3 layers and merged the shadows to the base layer rather than using 4 layers.

Without straying from the main topic and going into great detail about how we modeled and textured each component, that pretty much covers the use of Blender the pinball game.

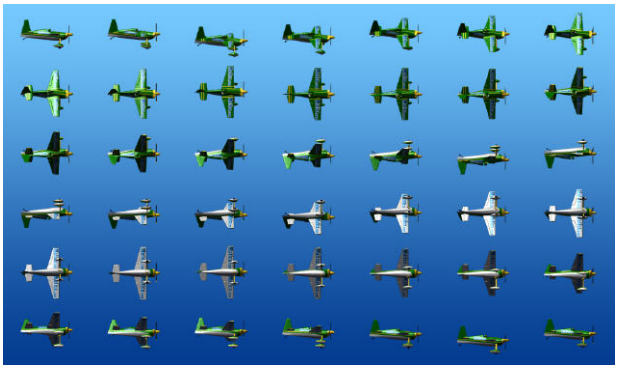
Matica Air Race

Matica Air Race was the first game we developed using Blender in our production pipeline. Therefore, it took much longer to produce the graphical assets than Pinball due to the learning curve.

What we wanted was essentially the same as for the pinball game, 3D sprites. The difference being that we needed an animated 3D sprite for the Aeroplane. So there was no real difference apart from rendering out an animation rather than a static scene.

We rendered the aeroplane doing a 360 degree roll, and rendered the propeller animation separately as we could get away with a lot fewer frames of animation for the propeller and put everything together in the game.

I'd modeled the aeroplane using very roughly sketched blueprints I'd drawn by hand. I did this while watching video footage from the Red Bull Air Race, so the final aeroplane had pretty realistic proportions. This would have looked perfect in an animation but didn't sit well when it was put into the game as it was too long and thin.



It's something that is difficult to get across in an article but when we played our first prototypes of the game it was very clear something wasn't right.

Having the model, animation, and textures waiting for us in Blender, we could scale the model on one axis and render again to have a fully animated shorter and faster version in minutes. We quickly had a more cartoon like aeroplane which suited the movement and look of the game.

We had planned for the aeroplane animation to be used as a guide to produce a vector drawing but it was so good we stuck with the rendered images. At the small size we needed for the game the raw render looked significantly better than our vector tests and there was very little overhead in terms of filesize.

By Alex Aylesbury

By Alex Aylesbury

We went on to model other items for the game; rings, air gates, blimps, hot air balloons, even the 02 Arena all of which were very quick to create. We got a little carried away and modeled Tower Bridge which took an age to model and texture especially as it was our first Blender project. We could have certainly drawn that quicker using traditional 2D imaging software.

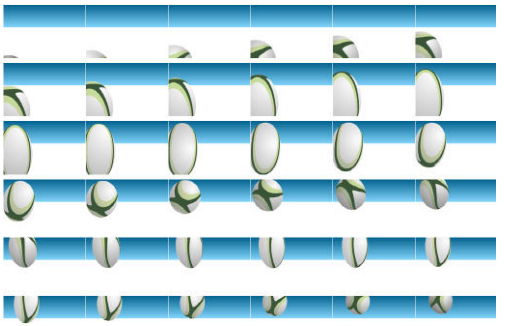
Many of our games and animations are not on our website as we are often hired to develop a bespoke project for a particular client. The games that make it up on our website are games we developed internally for fun and we often re-brand them for Clients which helps us to recoup our costs.

It's clear to see the benefits to using tools such as Blender in these cases. We can add a client's logo to the Air Race Plane, Balloon, or Blimp very quickly by changing the textures or even giving the textures to the client and letting them make the changes. We can totally re-brand our games in hours rather than days.

Rugby

Our Rugby game has existed in some form or another for a decade and all graphics were initially hand drawn using traditional 2D imaging software. This game gets a refresh every couple of years and during our current refresh we have replaced our hand drawn 3D assets (goal posts and a few other assets) using the same process that we have already outlined.

More interestingly, one of our clients wanted the game re-branded, and requested the addition of an intro to the game. They wanted a ball to fly onto the screen then go into a mode of spinning around, stopping, showing a message, spinning again, showing another message, etc.



We estimated 125 frames of animation would be needed (around 5 seconds) adding 2 or 3Mb to the game which until now was under 120k. We had to somehow add the intro without introducing this additional overhead.

The game would be promoted at events on plasma screens so had to scale up to 1920x1080 pixels. Not a problem for most of our games, as they use vector graphics, but quality would be lost if we used bitmap images at a lower resolution.

We used Blender in the same way we had used it in the 'Matica Air Race' game to model, texture, and animate the ball. Blender allowed us to export the animation as a series of .png files with transparency, and at the correct resolution to match the game.

To achieve a result that would add just 80k instead of 2-3Mb, and that would scale to any size without degrading we Rotoscoped the animation which has worked well for us on past projects.

We had actually planned to Rotoscope the 'Matica Air Race' Aeroplane but the original render results looked much better than our Rotoscope tests and as I mentioned earlier the file size was pretty good anyway.

The images I've included in this article are thumbnails of the final Rotoscoped vector images not the original rendered images.

We loaded the .png files we had rendered in Blender into Flash and traced the 125 frames by hand.

We played fast and loose for most frames only spending 2 or 3 minutes per frame as it gave a good visual result without being too accurate. The last frame we tried to get close to perfect as that was the only frame that would be in view for more than 1/25th of a second.

The resulting animation was just 86k and looked indistinguishable from the original animation when playing at full speed.

I've animated 3D scenes by hand in the past with just a few reference images and my imagination, and I've used this Rotoscoping technique working with Blender

animations as a template dozens of times. I'd need a very good reason not to use the latter as it saves much more time than it takes to create the Blender animation, and the result is a lot smoother.

You can stray from the original animation if you change your mind and want to get creative (which we do more often than not), but it's always there as a guide to help avoid your animation getting too jumpy, to give a guide to maintain the volume of your object, and as a guide to where the shadows and specular highlights fall.

It would be easy to go on describing more examples were we have used Blender in other creative ways in our projects, but hope what has been covered so far is of sufficient use to those planning similar projects.

Extra Resources

We have articles on our Website covering the making of a handful of our games, and have started recording game development video tutorials for our 'Game University' section.

http://www.matica.com - The Games

How we make the Games

Our new Games University section on how to make games

Alex Aylesbury

Professional game developer for 16 years, developing Web based casual games since 1996 and founder of Matica.com & Smilie.com.

By Alex Aylesbury

2.5D Games

By Paul Viraper

Introduction

In the last few years it's become more and more feasible to produce 3D browser based games, thanks to things like Papervision3D and advancements in Adobe's Flash Player, everyones favourite when it comes to online games and interactive content. The majority of personal computers have a version of Flash Player installed, with around 95% having the latest Flash Player 10, which is great because not everyone wants to have to download new plugins to play your game.

The thing is though, in modern console games we're all used to the latest graphics with their high polygon counts, real time lighting, high resolution textures, pixel shading and much more. But when it comes to 3D in Flash you're going to find things will soon start to lag the more complex you make your scenes.

This doesn't have to be a problem though, Here's the approach we use at Big Pixel Studios for our 3D games for you to think about. When starting out its good to have an idea of what's going to be involved in your game. How big is it going to be? How much is going on? How detailed does it have to be? ...etc.

In Big Pixel Racing (and our next 3D game which is top secret, so don't tell nobody else!) we've restricted the camera so it has a fixed view on the in-game world. The camera can pan and zoom in and out, but because we never rotate the view the 3D objects like buildings and trees can be kept relatively simple in terms of their geometry, and some are almost what you could call 2.5D.

To make up for the low poly count and lack of special effects we can use in Flash, we try to give our game a quite stylized look instead of going for photo-realism. For Big Pixel Racing we went with a 3D retro pixel graphics style, though thinking about it I would love to have seen photo realistic dogs driving cars around a dog version of Miami!

As the meshes we make are so simple a lot of what's going to get the message across is going to be the texture map, so I find that is the best place to start and the most important thing to think about.

Your texture map is going to be a guide to help you build your mesh (plus as in this case its going to be a simple pixel art style texture map so this is quickly becoming the easiest tutorial guide ever).

Though we are able to use lots of colours in our texture maps in Papervison3D, we have to remember the final look of our game. As our game has a retro pixel style it was good to use a limited palette of colours (this is a good thing to do in artwork in general) and colours that go well with the ground texture, The ground texture is something that comes together with the rest of the game objects in code, so I find it's useful to make a little sample of the ground texture on a 3D plane just for modeling with. This allows us to get an idea of how things might look in the finished game as we are modeling.

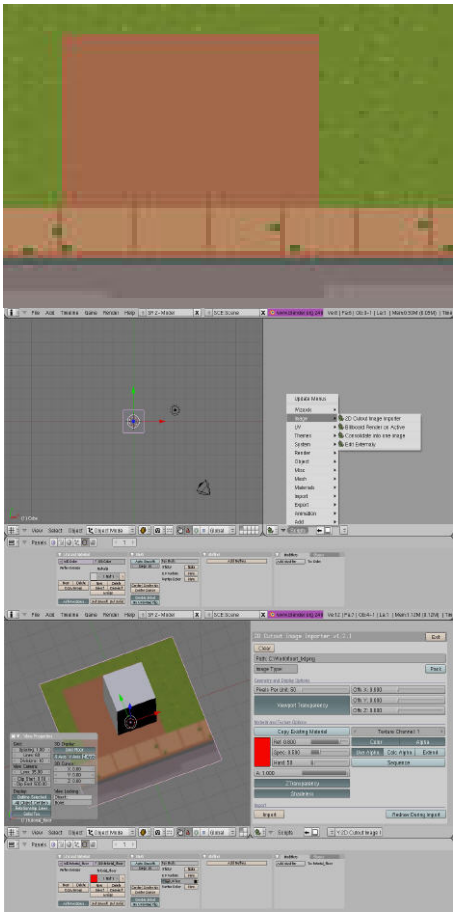
By Paul Virapen

In the bitmap editor of your choice make a small bitmap 50px by 50px. We choose such a small size because it will help with the pixel art. We're going to make a smallish square building here, we only need to make the texture for two walls and the roof of our soon to be very square pixel building, as we are going to reuse the walls on both sides of our mesh. In this little tutorial we are going to make a small cafe, its a nice cafe and the coffee is cheap, though they have been struggling recently as a large chain has opened down the road and people only want so much coffee in their lives, though we don't need to know this plus its a small pixel cafe and is not real.

Though lighting effects are possible in Papervision3D we didn't use them in Big Pixel Racing, so when making the texture map for the walls we made one darker to make it stand out from the other wall. This darker wall will be the one we use as the side walls to make our little pixel building stand out.

Once we've finished our texture map (it didn't take long) I like to double the size because 50px is small and you might drop it, or it might get lost down the back of the couch, or the cat might eat it and the vet bills for having small bitmaps removed from cats can be very expensive! Make sure your bitmap editor of choice hasn't resampled the image when you scale it up, we don't want fuzzy pixels. Save the new texture map as a png, you can give it a name like 'cafe_txt.png' because that sounds cool.

Now, before we open Blender there is something I must tell you. It was November and I was in the bathroom standing on the toilet hanging a clock on the wall when I slipped hitting my head on the sink. When I awoke I wrote this... which turned out to be the links for the 2 plugins that will help us in Blender. If you don't already have them you should download them now, but I bet you do because everyone does.

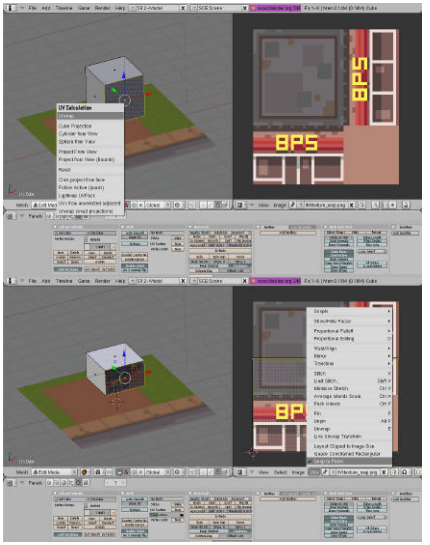


When you open Blender there's already a cube, lets call him Cubey. Don't bother renaming him I just mean from now on his name is Cubey, you can use it when you pass him in the street. We'll use Cubey as the mesh for our really simple building example.

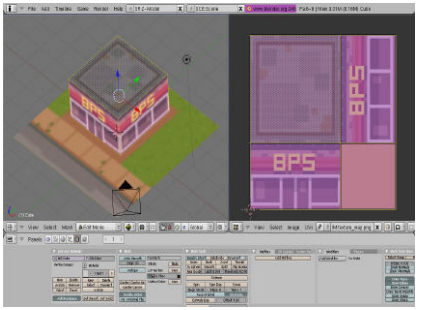
First lets' make some ground' as it were. We're using Blender 2.46 here. I like to split the main 3D area and make the right side the script view then go Scripts->Image->2D cutout image importer. Then we can import our floor texture sample so we can get an idea of the floor.

By Paul Virapen

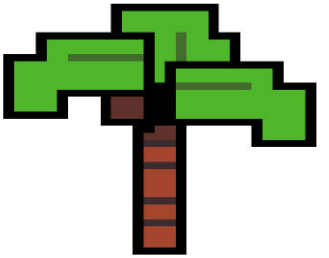
We don't have to worry really about making materials because this is all done on the Flash side, as long and we set up the UV coordinates with the texture map it should be okay. You can make a material in blender though if you want to do a test render, we do this to get an idea of how it will look by turning off the anti aliasing then rendering it from a camera view similar to the one used in the game. Setting up a camera like the in-game view is good when you want to see how your models will look.



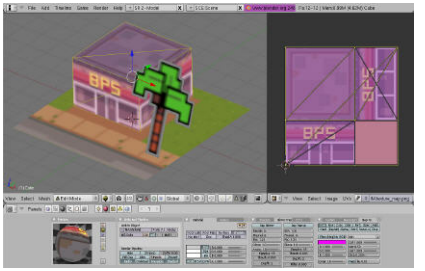
Grab Cubey and in edit mode lets select his bottom and delete it, we don't need to see Cubey's bottom today, no one does. Set the right view to UV/image editor then load the cool_txt.png. Then with Cubey minus his bottom, still in edit mode, grab a soon to be wall side and hit the 'U' key to get the UV drop down and then press Unwrap. Now in the UV/image editor window go to UVs->snap to pixel. Because our texture map is so small it will make life simple. Adjust the face-wall in the UV editor and make it fit with the wall on the texture map. Do this to all of Cubey's remaining sides one by one, adjusting it until it all fits nicely. That's it, that was really simple.

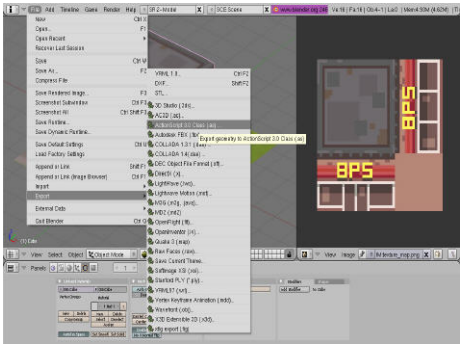


Adjust the mesh until you are happy with it then go back into edit mode and use convert to triangles so Cubey will export nicely.



For things like plants and trees we use 2D planes at slight angles almost parallel to the camera, remember if we don't rotate that camera no one will know. Then we have a nice and very simple little scene, you can make a few different objects in a similar way to make up street 'furniture' which can be reused.





When you want to export something, select it and go to file->export->actionscript 3 class. Make sure you've converted to triangles first though.

And remember not to let your cat eat bitmaps or meshes because they will if they can ■

Links:

http://wiki.blender.org/index.php/Extensions:Py/Scripts/Image/Import_2d_Cutout_Images

http://www.rozengain.com/blog/2008/01/02/export-your-blender-objects-straight-to-away3d-papervision3d-and-sandy/

By Paul Virapen

Infinite Skies

By Hubert Gizeskowiak (PM)

Introduction

"How about developing our own video game?" This is probably one of the ways your best buddy would try to convince you to spend all of your spare time with a never seen, endless, giant game development project called Azure: Infinite Skies. So yeah .. why not?

It has almost been an year since we started working on this and we've won a few members who occasionally helped us, mainly with coding. Currently we're 3 people working in our spare time on Azure and a few external helping out here and there. In the beginning we had to fight endless verbal battles about what the game should look like or what the gameplay should be about.

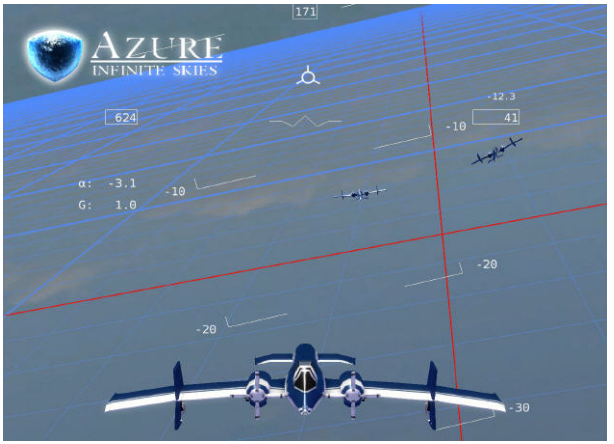
Some wanted a fantasy, others an action arcade game, but after a month or so we finally came to the decision to make an open source action flight simulator with elements of role-play games. The major part of the game should consist of flying around within the isles of Azure, an endless sandbox fantasy world, styled similar to our world during the 1940's - 1950's.

Because we didn't want to create another boring World War II shooter game and because just flying around would be annoying in long term, we added some special fantasy features like levitating Isles driven by a special mineral which makes things levitate. This mineral is also used for military and civilian purposes like giant flying ships, carriers or even man-made stations in the sky. Maybe some of you remember the huge carriers from the movie "Sky Captain And The World Of Tomorrow" - well, that's what we're aiming at. Also, inspired by Crimson Skies, all our aircraft are extraordinary, while staying authentic. Here we're looking pretty much into WW2 concept designs and experimental aviation. Besides all that technology fun, our motivation is led by the goal of creating a sad story and an unfamiliar atmosphere, where you question your own choices sometimes. This makes the game something special because it gives the player not only a yet not seen world to explore, but also a lecture of self-experience.

The second step was to choose the right tool-chain and libraries. Gimp was what we all knew for 2D and because it's free and cross-platform - -which is important as we all use different OSes- the choice was clear. For modelling we've chosen Blender, of course, as we had some positive experiences with it before already. Currently we're using it for almost everything visual in the game like aircraft, carriers, islands and so on. While staying open to other preferences of external modellers, our pipeline has been optimized mainly for Blender.

As game engine, we've chosen Panda3D after a long time of comparing, and it gave us everything we needed for the development. Unfortunately the BGE was, and still, is lacking a few things we felt were important. Panda3D is mature, free (BSD) and has many nice features, not to mention the great community. Same as Blender it's cross-plattform, written in C/C++ and allowing control through Python bindings. So yes, we code our game all in Python, and it has worked out very well thus far.

At the moment we're finishing ODE-based physics and collision detection, so the backend shouldn't need too long now. After that we'll focus on



adding more assets like planes and scenery. The story first needs some love before we think about implementation - we have some ideas all written down on our forums, but we're looking for a skilled storywriter to combine it all. As probably every ambitious open source project we're always open for new, skilled members, but we're also progressing on our own quite well. That said we'll probably release a first tech demo in the next few months where you can fly a plane around and see some nice effects.

For those interested in more info, we have a forum and wiki both hosted at SourceForge. All further info, like contact information and more, to the whole concept are written down there. If you have questions, you're very welcome to our IRC channel - info on the wiki.

Have a nice day,

Azure Development Team ■

By Hubert Gizeskowiak (PM)

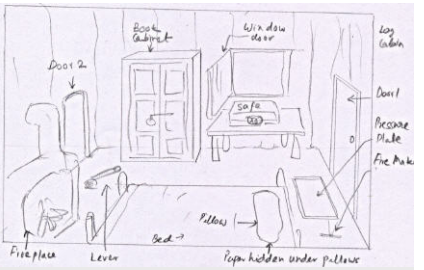
By Gaurav Nawani

Introduction

We at IronCode Gaming have been using Blender in our games for the past 6 years. Last year we released an adventure game titled "Pahelika: Secret Legends". It is basically a 2D game which contains simple point and click adventure dynamics.

Let me first get into how the game is developed and how we approach the artwork in Blender. Firstly since this game is completely 2D, we require all of our art assets as images and sprites. Based on the story locations in this game we have a set of stages and their corresponding levels set in a mystic environment.

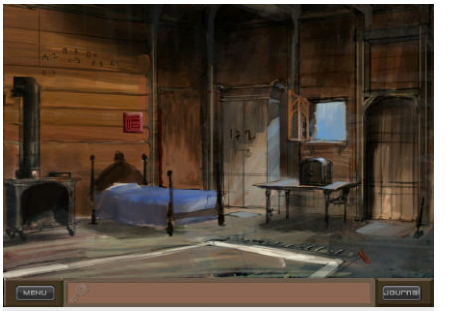
Documentation describing the scenes was developed first and a initial scratch of the concept by the game designer, which was then sent on to the concept artist for further development. After his initial penciling is approved, the concept artist prepared a final concept rendering in color. To keep the time short and expenses under control the artist was instructed to draw and render the bare minimum as to give enough ideas about color, tone and design for the 3d artist to work on.



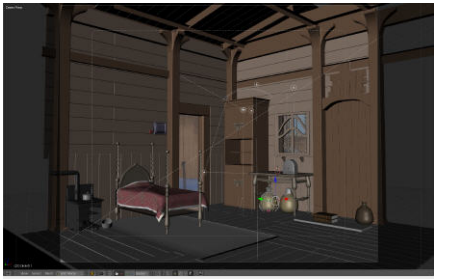
Scratch developed by game designer.

This finalized color concept art is now ready to be made into a 3D rendered scene using Blender. A 3D artist now prepares the props and assets. The final render of the scene and various interactive object renders were prepared to be included in the game.

In all of Pahelika there were about 30+ scenes that were rendered with Blender.



Concept Developed by the Artist



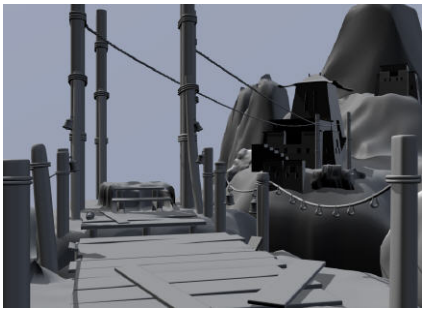
Concept prepared in 3d in Blender

Final Scene rendered in Blender

By Gaurav Nawani

Modeling

Since the scenes each contain various entities, it was a long job of recreating them all in 3D and preparing them for texturing.



Texturing

For most part the texturing is a mix of procedural or handmade 2d textures in Photoshop or Gimp. Some required a mix of both or even node textures.

Lighting

The trickiest part of the entire process was lighting. Realistic lighting could be achieved by using AO but it also increased the render time tremendously, given the number of small but detailed models in the scene. None of the images that were rendered used AO and doing so helped us in drastically reducing the overall time of the project. A normal AO render was anywhere from 15-20 minutes, while most renders we finally used were done in less than a minute on a quad core machine.

Since this is a one man team doing the 3D art it was decided to use the Blenders Lighting for most parts, so it was a mix of scan line as well as raytracing for all of the 30+ scenes.

Conclusion

The only gripe we had about Blender was its low quality of raytracer and of course the complete absence of Global Illumination and caustics. Technologies like these are best suited for our type of work, so we also spent time testing out Yafaray. It is excellent raytracer but it sorely lacks the integrated facility of BI meaning a longer learning curve in textures as well as the lighting setup. We were fairly satisfied with Blenders quality and also my ability in using it ;). We have high hopes from Blender 2,5 for the renderer and shader improvements that it can bring to us, hopefully before we release our next game.

Thanks for reading ■

Shootin Annie

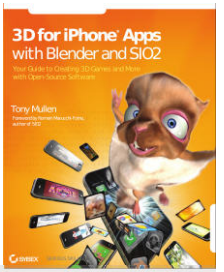
By Tony Mullen

Introduction

3D games for iPhone

Apple's iPhone/iPod touch platform has become one of the most exciting game development platforms around. The iPhone's innovative, high-resolution multi-touch screen, integrated accelerometer, network connectivity, and ultra-compact size have together effectively revolutionized palmtop gaming and application interfaces. In addition to a great device, Apple has introduced a development model that enables 3rd party developers to create programs that run on the device, and created a complete all-in-one solution to make developing and marketing the apps as straightforward as possible.

The iPhone platform includes an implementation of the OpenGL ES graphics library for mobile devices, making it possible to create top-quality 3D content for the device. Coding directly for OpenGL ES requires advanced skills in graphics programing, however. For most of us Blender users who want to create 3D games a higher-level, more intuitive game development engine is desirable. Fortunately, there are several such game engines available as open source software. I chose SIO2 Interactive's SIO2 game engine to work with because of its excellent support for working with Blender assets. Another well-known alternative is the Oolong game engine, which also supports working with Blender 3D content.



3D for iPhone Apps with Blender and SIO2 is available now on Amazon

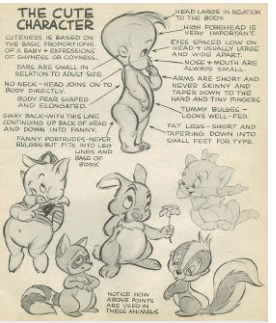
In this article, I'll take a brief look at some aspects of the creation of my own commercial 3D iPhone game Shootin' Annie, created with Blender and the SIO2 game engine. The game itself is available on the iTunes App store for $0.99, but with this issue of BlenderArt magazine, you can download the .blend file containing the main character assets for the game, which is freely distributable for educational purposes. Unfortunately, it's not possible to give a full introduction to working with the SIO2 game engine and Blender to create iPhone games. However, a complete introduction to creating 3D content for the iPhone with Blender and SIO2 can be found in my new book 3D for iPhone Apps with Blender and SIO2: Your Guide to Creating Games and More with Open Source Software. The book is slated to hit shelves February 15th, 2010, so it's likely that by the time you read this, it is already available. If you have any trouble following the Blender parts, you'll find much more information about the methods I describe in my other books.

The Shootin' Annie splash screen on the iPhone simulator

By Tony Mullen

My goal with Shootin' Annie was to create the simplest possible character-based 3D game I could. Character creation is one of my favorite things to do in 3D, so I knew I wanted a character-based game, but I didn't want to saddle myself with the need to create the kind of complex gameplay that many 3D character-based games demand.

The Annie character herself was inspired by a Blender-made video I saw on the Japanese video website NikoNikoDouga featuring a miniature (chibi) version of the virtual popstar icon Hatsune Miku. I found the chibi design of that character to be very appealing, and I wanted to create a character with similar appeal. Although the style is of course similar to the Japanese kawaii characters that inspired it, most of the design principles extend all the way back at least as far the early animation work at Disney and MGM, and I made explicit reference to Preston Blair's classic book Animation to get the dimensions right (that book is available in its entirety from The Animation Archive at www.animationarchive.org).



"The Cute Character"

The Annie character includes most of the classic signifiers of cute: outsize head; big eyes; tiny mouth, nose, and chin; small hands and feet; and little or no neck. With these dimensions, a character will look cute even when she's firing six-guns in a black ten-gallon hat and boots. I liked this contrast between the cute and the badass, and once I hit upon this character the rest of the game concept soon followed.

With respect to gameplay, my inspirations ranged from Tetris to the early cell-phone Snake-style games to just popping bubble wrap. I wanted something super simple, but still a bit compulsive. What I came up with was a simple old-school shooting gallery game with a few twists. Instead of going straight across the screen, the targets rotate around the Annie character in the middle of the screen. To hit the targets, the player taps, double-taps, or triple-taps on the target depending on the number of white rings in the target. This causes the Annie character to spin around and shoot the target.

This reportedly Blender-made animation of Hatsune Miku and friends was an inspiration for the design of the Annie character. You can see the animation on YouTube .

Missing a target results in losing a bullet. The iPhone's multi-touch functionality is used as well. When the bullet reload icon flies around, tapping it with one finger will fill up one gun (6 bullets) and tapping it with two fingers will fill up both guns. The number of targets, their speed, and the variation in the number of rings on each target increases as you progress, as does the complexity of the targets' design.

By Tony Mullen

Modeling

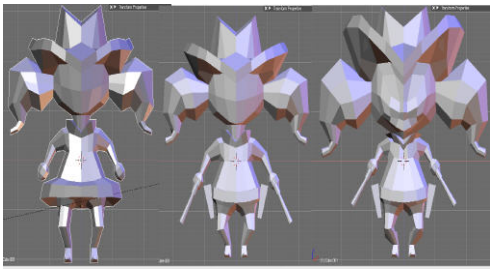
The model was created freehand in Blender, using all quads. It's much easier to work with quads when modeling than tris, so converting the model to tris is something I only do when I'm finished modeling. I used the mirror modifier to model but no subsurfing because the model was intended for real-time (game) rendering. I find modeling in Blender almost as intuitive as I do drawing with a pencil, so for non-realistic subjects I often model without a reference drawing. In this case, the added flexibility of modeling and designing on the fly

Game play in Shootin' Annie is simple but gets challenging as things speed up.

was helpful because I could change the dimensions of the model quickly and easily to see how various proportions worked. For example, I started the model with much less exaggeration of the head size, but found that a more drastic exaggeration suited the character better.

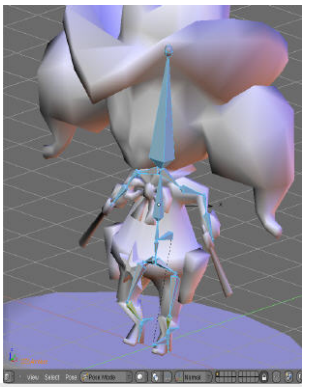
Rigging

The armature for Annie uses 24 bones. This is about as complex as I would dare to make an armature for use in SIO2. If possible, the armature should use as few bones as possible, but since the other demands of the game were fairly minimal, I figured I could get away with a heavier armature here. The reason bones can slow you down is that each bone is converted to a vertex group in SIO2, and deformations are based on the vertex groups. More vertex groups take up valuable processing resources, which are at a serious premium on mobile devices.



Several points in the evolution of the character's proportions.

A nice advantage of working this way, however, is that SIO2 can handle all kinds of deformations on the mesh in an action. You can use FK posing, IK posing, even Lattices and Mesh Deform modifiers and the mesh will deform correctly in SIO2. The only thing SIO2 cares about is the location of each vertex in space during the action.

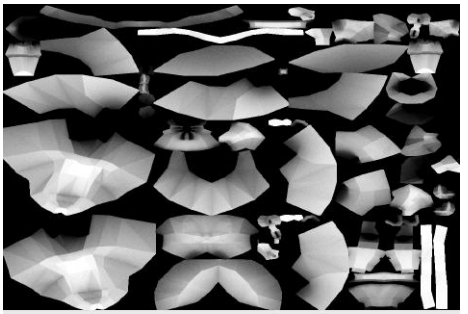


The Shootin' Annie armature is about as complex as you want to get for the purposes of an SIO2 game.

By Tony Mullen

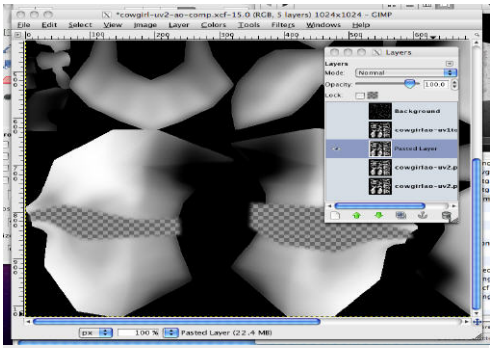
Texturing

I used multiple UV layers and texture baking in a crucial way to texture this model. If you're not familiar with using multiple UV layers and baking textures from one to the other, please refer to my book Mastering Blender for an in-depth tutorial on how to do this. This is a very powerful feature.

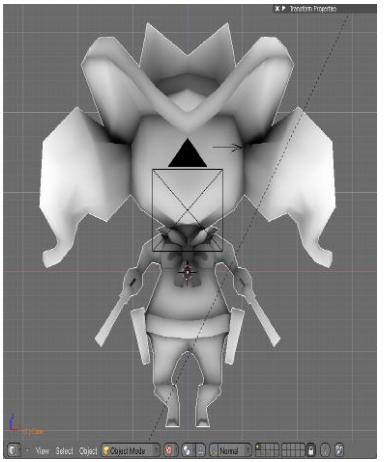


The first AO bake, without blurring.

First, I baked ambient occlusion to a UV texture. I wanted to soften the effect by blurring it slightly in GIMP, but when I did this, the blurred seams showed up on the mapped texture. To counter this, I re-seamed the model, baked the original blurred AO texture to one image file, then re-baked AO to another image file, and blurred that file in GIMP in the same way I'd blurred the original AO texture.



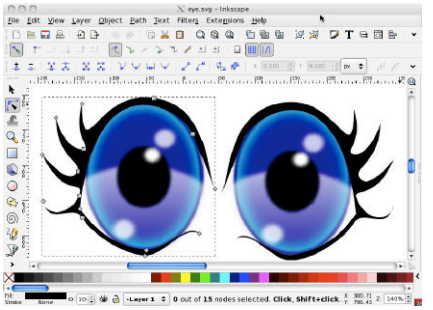
Compositing differently mapped AO bakes to get rid of seams.



The model with the baked AO texture applied

I then combined the two baked images to conceal the unwanted seams and used the resulting image as my final AO texture. The colors of the clothing were created directly with Blender's texture paint functionality, on their own image, which was baked to a single file with the other textures at a later stage.

The manga-style eyes I created in Inkscape. I stuck those to the face using an Object Mapped texture and an empty, in the standard decal-style texturing manner.



Manga-style eyes in Inkscape.

All three of these textures, the AO texture, the color texture, and the eyes decal texture I associated with a shadeless material, then I baked the full render to yet another UV texture, which included all of the texture information. I painted the nose and mouth onto that texture directly using GIMP, using a Kewpie doll as a reference.

Animation

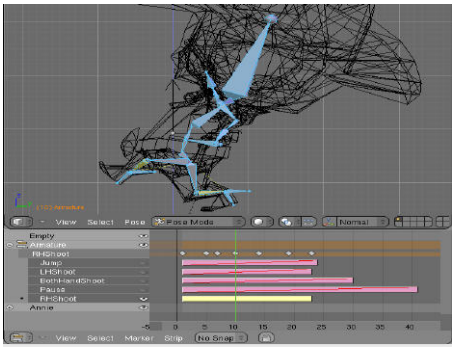
SIO2 can work with Blender animation (Ipo) curves and with actions. For character animation, you use actions. I created a rest action, where the character is just swinging her guns loosely, a jump action, where she jumps up in the air for the spinning movement in the game (the spinning itself I programmed separately in SIO2), and shoot actions for both right and left hands.

Working with SIO2

Developing with SIO2 is much simpler than programming directly in OpenGL ES, but it also bears little resemblance to the kind of programming you'd learn in most introductory iPhone programming books.

For ordinary applications, the main language used for programming on the iPhone platform is Objective C, an object-oriented variation of the C language. In addition, special tools such as Apple's Interface Builder enable iPhone programmers to create sleek interfaces quickly in a drag-and-drop, WYSIWYG fashion. This is what you'll learn in a standard introduction to iPhone programming. Programming with SIO2 is for the most part a completely different story.

With SIO2, you'll hardly touch Objective C except to edit or comment out the odd line of code here and there, and most of the programming you'll do will use simple C control structures within the context of readymade SIO2 templates. It will definitely help you to have some background in C, C++, or other C-like programming languages (an understanding of pointers is especially helpful), but no specialized programming experience is necessary to get started.



Animated actions in the Action editor.

I designed Shootin' Annie along the lines recommended by SIO2 Interactive in the tutorial games available on the SIO2 Interactive website, www.sio2interactive.com. In particular, the Meditation Garden game tutorial package shows how to organize assets within the SIO2 game environment for complex games.The basic workings of SIO2 are all explained in my book, but for more advanced projects, the SIO2 tutorial games are really invaluable. If you're just getting started with SIO2, there are also freely available tutorial videos and code packages available on the website. You can study directly from these or use them to augment your study from the book. For Shootin' Annie the game assets and necessary values are all held in a data structure I defined called SAGAME that is initialized soon after the launch of the application, outside of the main render loop of the game. In this way, all the SIO2 objects and resources needed can be accessed from that data object at any time during the execution of the game.

By Tony Mullen

The SIO2 resource data type is also crucially important in the design of the game. This data structure acts as a resource manager and handles the treatment of resources such as models or widgets. Any time you have a collection of objects that need to be cycled through and dealt with in some way, for example the collection of targets in Shootin' Annie, it is a good idea to give them their own resource manager and make that resource manager active when you want to deal with those specific objects.

For example, the collection of targets were imported using a separate .blend file, and they are handled by their own associated SIO2 resource manager.

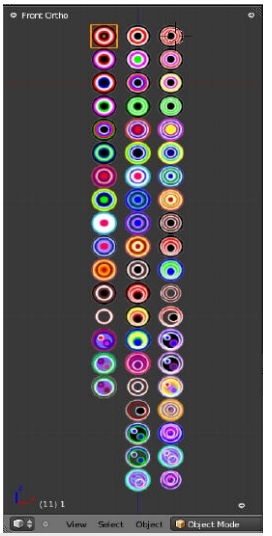
The way to bring 3D assets into SIO2 is to use the SIO2 Exporter Python script to export them from Blender to a special file format called a .sio2 file. I worked with multiple .sio2 files for Shootin' Annie. The book discusses exporting Blender files to the .sio2 format, but the .sio2 file format can also be used to bring other resources into the SIO2 environment, such as widget graphics. It's incredibly simple to do this. All you need to do is put the resources you want into a directory, zip the directory, and rename it from .zip to .sio2.

You can then access these files from within Xcode just as the book describes doing with 3D assets.

Working with Apple

Working with the iPhone SDK and Apple's various protocols and environments is a challenge in itself that many Blender users may not be accustomed to. Unlike the freedom-loving, open-source Blender Foundation, Apple runs a very tight ship, for better or for worse. Given the challenge of creating a lucrative, stable, and secure environment for developers on a mobile phone platform, it's hard to fault them for how they've gone about it.

Open source purists will probably not feel comfortable developing for this platform, but then again, until recently there were few if any open source options for mobile phones available, and most platforms did not allow 3rd party development at all, so the iPhone platform should be considered in this context. Hopefully Google's Android will open up lucrative opportunities for developers in the future.



The blend file contains the bulleyes

In any case, developers should be prepared to jump through some hoops to get their work made available on iTunes, beginning with owning a recent Mac. The iPhone SDK, including the iPhone Simulator, the Xcode IDE and a variety of other tools is available for free, so you can test the water with no expense, but to program for an actual device or to make your game available to others on iTunes requires a $99 membership to the iPhone Developer Program.

Once you're a member, there are procedures at every turn to ensure a secure and controlled developing environment. While these can be a bit of a hassle, the iTunes service runs remarkably smoothly once you've got all the formalities out of the way. When you've done this, you'll have the satisfaction of having your own game on the market for one of the hottest platforms around, and if you're lucky, you might make some money at it as well ■

By Tony Mullen

Mii School

By Moisés Espínola

Introduction

Mii-School is a 3D school simulator developed with Blender and used by psychology researchers for the detection of drug abuse, bullying and mental disorders in adolescents. This work was financed with a research project from the Spanish Ministry of Health and Consumption and developed in the University of Almería (Spain). The school simulator created is an interactive video game where the players (in this case, the students) have to choose, amoung 17 scenes simulated, the options that better define their personalities.

Different quality versions of Mii-School have been created, so the execution of the program can be adapted to the technical characteristics of each specific computer as closely as possible. So if Mii-School is executed on a latest-generation PC, it can use the highest-quality version to enjoy better viewing (XGA 1024x768 resolution). However, if the graphic card cannot reproduce this visual quality with sufficient fluency, other lower-quality versions can be executed (SVGA 800x600 or VGA 640x480 resolutions).

Development methodology with Blender

I chose Blender as my Mii-School development tool because this software incorporates all the functionality that I need to make a video game from beginning to end: 3d design, advance illumination and texturing, characters animation, Blender game engine, programming with Python, etc. Besides, Blender has the most important property that is most wanted in software: it is open source. Blender incorporates almost the same functionality as 3D Studio Max (whose license costs more than 3.000€) and Virtools Game Engine (whose license costs more than 9.000€), and Blender is free.

Mii-School development process with Blender was divided in three different stages:

a) 3D design, illumination and texturing

During this first stage, the 3D meshes were created with Blender to represent the virtual scenarios and characters that appear in the scenes. The Mii-School simulation takes place in different scenarios: the school playground, the classroom, the main character's home, a park... Each scenario is divided into several zones (for example, the home has several rooms, the playground has an area with benches, another for sports, etc.), each zone is used for a

different scene and the user does not get the impression that the scenarios are repeated. A total of 30 characters were designed in Blender: several male and female students of different races and social conditions, the teacher who is teaching in the classroom, the student's parents, etc. All of the Mii-School 3D models were created using basic and advanced design techniques included in Blender:

extrude, split, merge, etc. and finally, smoothing filters were applied for smoother surfaces without overly increasing the number of polygons (set smooth).

For proper illumination of the scenes, a sun type global lighting illuminates above the main student on whom the simulation action always focuses, so that the objects and characters around him are always properly illuminated. Some spotlights were also used to increase illumination on certain places.

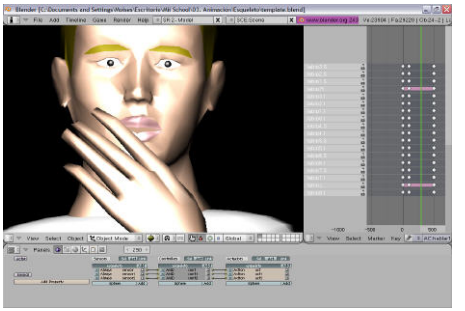
Texturing was also done during this first stage using the UV mapping technique and the Blender materials editor, applying good quality images to the 3D models to increase realism of the scenes and characters.

b) Animation

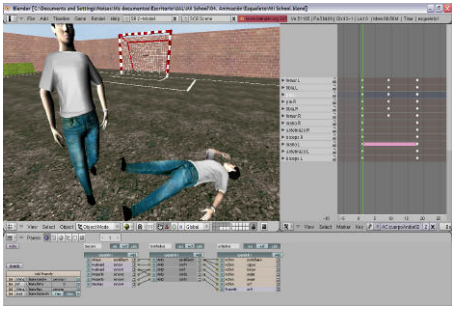
Once the 3D models were designed, I proceeded to the animation stage by first creating skeletons associated with the character's 3D meshes, then capturing bones movement using a series of intermediate poses with the aid of a technique called inverse kinematics.

Character's animations were divided into two large groups: body and facial. Body animations affects the character's whole body. I implemented several types of body animation in which the characters perform some action: walking, sitting, hitting, threatening, lying down, fighting, smoking, drinking, etc.

In order to increase the expressiveness and realism of facial animations, the number of polygons and level of detail have been increased noticeably in eyes and lips, areas that most influence the facial gestures. So if one character is threatening another in a certain scene, his face expresses aggressiveness. If, on the contrary, a character feels threatened, his face shows fear.



c) Blender Game Engine and Python programming



When the characters were created and the corresponding animations implemented, I started the coding stage using the Python programming language and the Blender Game Engine. Over 6,000 source code lines have been implemented in Python to simulate the action of the 17 scenes. This source code is the Mii-School kernel. It specifies all the details necessary to compose the scenes and also captures all of the information provided by the student. The Mii-School kernel is interpreted by the Blender Game Engine.

By Moisés Espínola

By Moisés Espínola

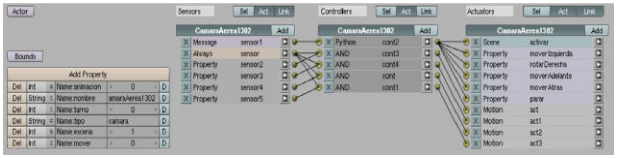
The source code specifies aspects such as what characters intervene in each of the scenes or which scene is going to be developed in this moment. In order to achieve better performance during program execution in the simulation of a scene, only the characters and objects that are going to be viewed by the camera are shown, the rest are hidden.

The source code that I have programmed in Python follows the instructions of a deterministic finite automata to find out in what state of simulation Mii-School is at any given time. It then activates the corresponding camera, body animation, and facial expressions of the character who is speaking at the moment. The following text shows part of the source code for the first scene in Mii-School. This code specifies the person speaking in this moment, his basic body animation, the facial expression (mouth and eyebrows), the active camera and the dialogue that is visualized on the screen. Finally, the complete scene is activated by sending all of the data to the Blender Game Engine.

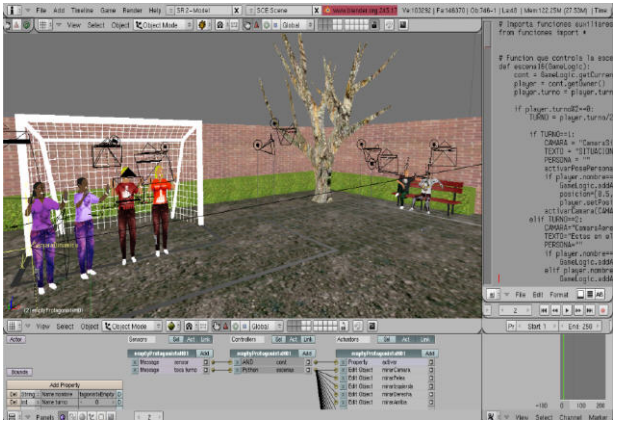
```
...
elif STATE==6:
CHARACTER="baddie01"
BASIC ="bodyHandsHips"
MOUTH ="mouthNormal"
EYEBROWS ="eyebrowsNormal"
CAMERA="cameraCharacter01"
TEXT = " We want to play handball, but we have not a ball ...”
activateScene(CHARACTER, BASIC, MOUTH, EYEBROWNS, CAMERA, TEXT)
elif STATE==7:
CHARACTER ="baddie02"
BASIC ="bodyHandMouth"
MOUTH ="mouthLaugh"
EYEBROWS ="eyebrowsBaddie"
CAMERA ="cameraCharacter01"
TEXT = "Perhaps we could use ... your head as a ball !!"
activateScene(CHARACTER, BASIC, MOUTH, EYEBROWNS, CAMERA, TEXT)
…
```

To simulate Mii-School, the Blender Game Engine not only executes the Python source code that I have developed, but also takes into account a series of events associated with the dynamic objects in the scenes, especially characters (both their 3D mesh and the skeleton) and cameras (with their corresponding dialogue box, which also has associated events). Each event is divided into 3 parts: the sensor (that receives information, for example, pressing a key or a specific signal), the actuator (which performs actions on objects, like animating a character or moving a camera) and the controller (which connects the sensor to the actuator).

In the Mii-School project, there are over 900 basic objects with associated events. Some are complex objects such as a character's skeleton or its 3D mesh, others are simpler, such as the dialogue box for a specific camera. Each object has an average of 10 events of its own that regulate their functioning. For example, in the cameras an event can indicate a movement, in a character an event can activate an animation.



The Blender Game Engine also takes other secondary details into account, such as which scenes the main character's cigarette or bottle should be seen or hidden, or what color the conversation box should be depending on the character who is speaking at that moment.



To summarize, during the execution of Mii-School, the Blender Game Engine must synchronize around 10,000 different events in real time, activating only those that are necessary right at that moment of the simulation and discarding the rest. Apart from this work, it is also in charge of rendering and illuminating the scenes in real time as well as capturing information introduced by the user.

Simulated scenes in Mii-School

During the execution of Mii-School, the student watches a total of 17 interwoven scenes studying different aspects of his behavior related to bullying, drug addiction, family life, capacity for attention in class and integration in social groups.

There are a total of 5 scenes that study bullying. In some of them, the student is bullied by his schoolmates and in others he becomes the bully. In this way bullying can be studied from several perspectives. Some concrete cases are also studied, for example, the reaction of the student to the explicit violence of a physical aggression in the schoolyard to see if he is a mediator or, to the contrary, violent. For each bullying scene, the student can select from a series of choices that always follow a general pattern: feeling indifferent to the bullies, protesting to them, responding ironically, running away in fear, facing up to them or feeling ashamed.

By Moisés Espínola

During the 6 drug addiction scenes, the drug offered becomes gradually more dangerous. In the first scene, the student is tempted by his schoolmates to smoke in the school playground during recreation. In the second scene, he is invited to drink alcohol while eating a pizza at a friend's house. Afterwards, he is offered a joint of marijuana in a park. In the last two scenes, the risk is upped further when his friends offer him much more dangerous drugs, such as cocaine or ecstasy. The choices that the student can choose in the drug addiction scenes also follow a general scheme: usually use, refuse to try the drug, advise friends to stop taking it, use occasionally or leave because he feels uncomfortable.

The student's relationship with his parents is also studied in three scenes: one scene checks the father's behavior, another, the mother's attitude, and a third scene both parent's behavior. In the first two scenes the student gets home much later than the hour agreed upon and checks whether the father or the mother scold him, threaten him or, on the contrary, are indifferent to his undisciplined behavior. In the third scene, the student gets home after school and feels anxious because he has problems with his studies and the simulation checks whether his parents become involved in his problems or are indifferent to them.

By Moisés Espínola

each one of the scenes and other data of interest, is stored in a web page format for later viewing and analysis by the psychology researchers.

There are other scenes where personality-related problems and the student's mood, attention in class, beliefs and integration in social groups are checked.

Final results and future works

The total simulation of Mii-School's 17 scenes lasts approximately 25 minutes. The student can choose his sex at the beginning of the simulation to personalize the animations to his own gender. All the information introduced by the student, as well as the choices selected in

Background music has been added in the scenes and audio in the conversations to achieve more realism. Proper movement of the cameras during the transition of scenes is also carefully made so the simulation has the quality of real movie.

As a future enhancement of the Mii-School program, the simulator is going to be translated into different languages ■

By Moisés Espínola

Moisés Espínola

Website: http://www.ual.es/personal/moises.espinola

Fullmetal Soccer - by Bernhard Bauer

Zero Ballastics -by Bernhard Bauer

Ranger - by Mark Wowk

Riverling - by Mark Wowk

Atlantis Return (Game) -by Alvaro Corts

Pandora floating islands -by Victor Jesus Acevedo del Real

Quarantine -by Seba

Sweet little soilder -by Zoltan Miklosi

SnowScene -by LightningIsMyName

Want to write for BlenderArt Magazine?

Here is how!

1. We accept the following:

- Tutorials explaining new Blender features, 3dconcepts, techniques or articles based on current theme of the magazine.
- Reports on useful Blender events throughout the world.
- Cartoons related to blender world.

2. Send submissions to sandra@blenderart.org. Send us a notification on what you want to write and we can follow up from there. (Some guidelines you must follow)

- Images are preferred in PNG but good quality JPG can also do. Images should be separate from the text document.
- Make sure that screenshots are clear and readable and the renders should be at least 800px, but not more than 1600px at maximum.
- Sequential naming of images like, image 001.png... etc.
- Text should be in either ODT, DOC, TXT or HTML.
- Archive them using 7zip or RAR or less preferably zip.

3. Please include the following in your email:

- Name: This can be your full name or blenderartist avtar.
- Photograph: As PNG and maximum width of 256Px. (Only if submitting the article for the first time)
- About yourself: Max 25 words .
- Website: (optional)

Note: All the approved submissions can be placed in the final issue or subsequent issue if deemed fit. All submissions will be cropped/modified if necessary. For more details see the blenderart website.

Upcoming Issue 'Theme'

Issue 27

"Shadow Showdown: Achieving Light/Shadow Balance"

- Lighting game levels and characters, images, animations
- Atmospheric lighting
- Nodes/compositing for great lighting effects
- Coloured shadows
- Ambient Occlusion
- Lighting groups
- Textured lighting/shadows

Disclaimer